2011 NDIA SYSTEMS ENGINEERING CONFERENCE

OBJECT BASED SYSTEMS ENGINEERING

Pradeep Mendonza
Systems Engineering Group
US Army - TARDEC
Warren, MI

John A. Fitch
Science Applications
International Corporation
Sterling Heights, MI

ABSTRACT

Historically Systems Engineering (SE) practitioners have focused on producing document-based artifacts to relay SE knowledge to stakeholders and developers. More recently a large part of the community has moved towards using view-based artifacts; e.g. DODAF to visually communicate SE knowledge. Documents and views are simply containers that hold objects. SE knowledge is comprised of sets (classes) of objects that are related to each other. By directly managing these objects, containers can be reproduced as desired.

This paper proposes a shift in the focus of SE from documents and views to objects. We are not proposing a new SE methodology, but rather a set of principles to integrate the information created by multiple, diverse SE methods.

Object based systems engineering is based on the following principles.

- *Map all SE knowledge to object classes and subclasses*
- *Refine this information architecture against multiple SE methods to make it as lean as possible (maximize cohesion, minimize coupling).*
- *Create all objects in context (within a hierarchy appropriate to its class)*
- *Define each object as a set of lean attributes and relationships (avoid free-form text).*
- *Strive for zero redundancy (store a single instance of an object; visualize in many ways).*
- *Maintain continuous traceability as knowledge is derived.*
- *Capture the precious and transient logic behind this knowledge derivation.*
- *Leverage the relationships between objects to proactively manage change.*
- *Maintain continuity of objects across system/product life cycles and phases.*
- *Harvest and reuse knowledge patterns for each class of object*

These principles may be applied to a diverse set of SE environments to simplify SE tasks, reduce overlapping efforts and information silos, foster the insight that leads to innovation, improve solution quality and accelerate development.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
17 OCT 2011	**Journal Article**	**17-10-2011 to 17-10-2011**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Object Based Systems Engineering	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
Pradeep Mendonza; John Fitch	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army TARDEC ,6501 E.11 Mile Rd,Warren,MI,48397-5000	**#22370**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
U.S. Army TARDEC, 6501 E.11 Mile Rd, Warren, MI, 48397-5000	**TARDEC**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
	#22370

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Submitted to 2011 NDIA SYSTEMS ENGINEERING CONFERENCE.

14. ABSTRACT

Historically Systems Engineering (SE) practitioners have focused on producing document-based artifacts to relay SE knowledge to stakeholders and developers. More recently a large part of the community has moved towards using view-based artifacts; e.g. DODAF to visually communicate SE knowledge. Documents and views are simply containers that hold objects. SE knowledge is comprised of sets (classes) of objects that are related to each other. By directly managing these objects, containers can be reproduced as desired. This paper proposes a shift in the focus of SE from documents and views to objects. We are not proposing a new SE methodology, but rather a set of principles to integrate the information created by multiple, diverse SE methods.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**13**	

Background

Systems Engineering is a knowledge-based process. Its success depends on timely, efficient and effective knowledge capture and sharing among a diverse set of system stakeholders, contributors and implementers. Historically, document-based artifacts have been the primary method of knowledge sharing among Systems Engineering team members. However, a document-centric paradigm works against timeliness, efficiency and effectiveness by:

- Holding emerging system knowledge hostage until the next document review cycle
- Triggering the replication of much potentially-common data between documents
- Capturing information in large, free-form text paragraphs that fail to separate and singularize important system data elements. Documents fail to clearly distinguish object, attribute and relationship data blended within large paragraph “blobs”. This leads to high variability in their content, format and interpretation among multiple contributors or users; unnecessarily adding to the ambiguity associated with the system model.
- Focusing engineers on writing tasks, not thinking tasks (e.g. writing requirements instead of defining them). If it's true that "The medium is the message” and "You get what you ask for", asking for document prose works against lean and effective Systems Thinking.
- Conflating document structure with system decomposition hierarchies.
- Forcing users to create, maintain and synchronize redundant copies of data elements if they wish to view this data in different contexts and for different purposes.
- Creating unnecessary manual effort to maintain unique requirement identifiers and traceability matrices.

Documents were the best possible approach when all that existed was a manual typewriter.

The trend towards Model-Based Systems Engineering (MBSE) has shifted the emphasis of the Systems Engineering community away from documents towards view-based artifacts. These graphical and tabular views (whether SysML or DoDAF-based) are a welcome step toward a more precise method of capturing and communicating Systems Engineering knowledge.

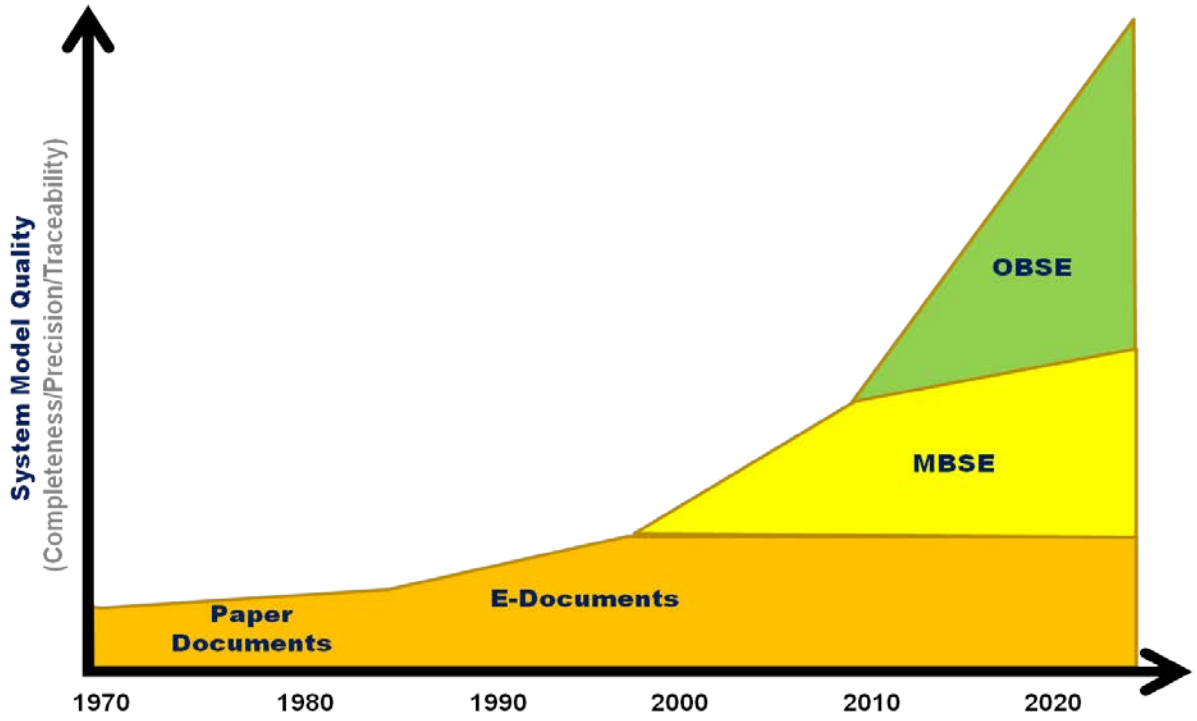
However, views also are less than the ideal focus of Systems Engineering:

- Views don’t offer full coverage of all classes of SE knowledge. They don’t capture the full decision and derivation trace that enables proactive impact/change analysis and reuse of knowledge across system life cycle phases. [1]

- Views focus engineers on drawing diagrams or populating tables. While this is much better than a document-authoring paradigm, the views may still become an end in themselves rather than the natural byproduct of continuous and effective Systems Thinking.
- Views are often populated (e.g. drawn, compiled) after-the-fact from other sources. While they contain objects and relationships, this data is often a copy of the original/master that is stored elsewhere. This increases the effort/cost/time required to maintain a consistent and traceable Systems Engineering knowledge-base.

Both documents and views are necessary, but not sufficient representations of Systems Engineering knowledge. They are simply containers that hold the objects that ultimately represent the essence of a system. The value created by Systems Engineering lies primarily in these objects, not the containers that deliver them.

FIGURE 1: Evolution of Systems Engineering Practice



MBSE proponents and initiatives clearly have Object-based System Engineering as their end goal, but the state of the practice lags the vision. The goal of this paper is to help accelerate the realization of Object-based System Engineering in the everyday practices of the defense community. [2] [3] [4]

Object-based Systems Engineering Concepts

Object-based Systems Engineering is based on the simple concept that SE knowledge is comprised of sets (classes) of objects that are related to each other. The essential elements of Systems Engineering can be represented by ***objects*** that are comprised of and defined by ***attributes*** and associated through ***relationships***. These objects can be grouped into logical classes using an affinity process.

Software Engineering has decades of experience in mapping diverse types of knowledge to Entity (Object)-Relationship-Attribute (ERA) models. There is nothing about Systems Engineering data that makes it impervious to similar treatment. It is certainly possible to create a comprehensive information architecture that captures Systems Engineering knowledge, but "possible" does not imply "easy".

There is no "right" information architecture for Systems Engineering; any information architecture is the result of design decisions in which alternative data models are evaluated against the needs of stakeholders (SE process owners, participants and customers). An information architecture could be complete (i.e. have a place for all required information) but still be inefficient (hard to maintain) and ineffective (hard to understand or hard to use in the SE tasks that it supports). The information architecture may vary for different domains, e.g. the level of system context; software vs. hardware emphasis, or life cycle phase.

If a high-quality (complete, simple, efficient, effective, scalable) information architecture is defined to support the Systems Engineering process, an organization can reap the following benefits:

- Reproduce artifacts (documents, paragraphs, diagrams and tables) by automated rule-based assembly of sets of objects.
    - Populate paragraphs by the concatenation of object attributes and relationships.
    - Populate diagrams from objects (nodes) and their relationships (lines).
    - Populate table rows (objects) and columns (attributes, linked objects).
- Shift the focus from artifact (document, view) reviews to object quality. Reduce the defects that escape into the final artifact.
- Analyze diagrams and tables using rule-based exception reports. Highlight missing, incomplete or inconsistent data. Improve the quality of an artifact by controlling its inputs, not by attempts to "inspect-in" quality at the document or view level.
- Eliminate the variability between the actual system model (requirements, design, architecture) and the views used to communicate the model. This variability could result

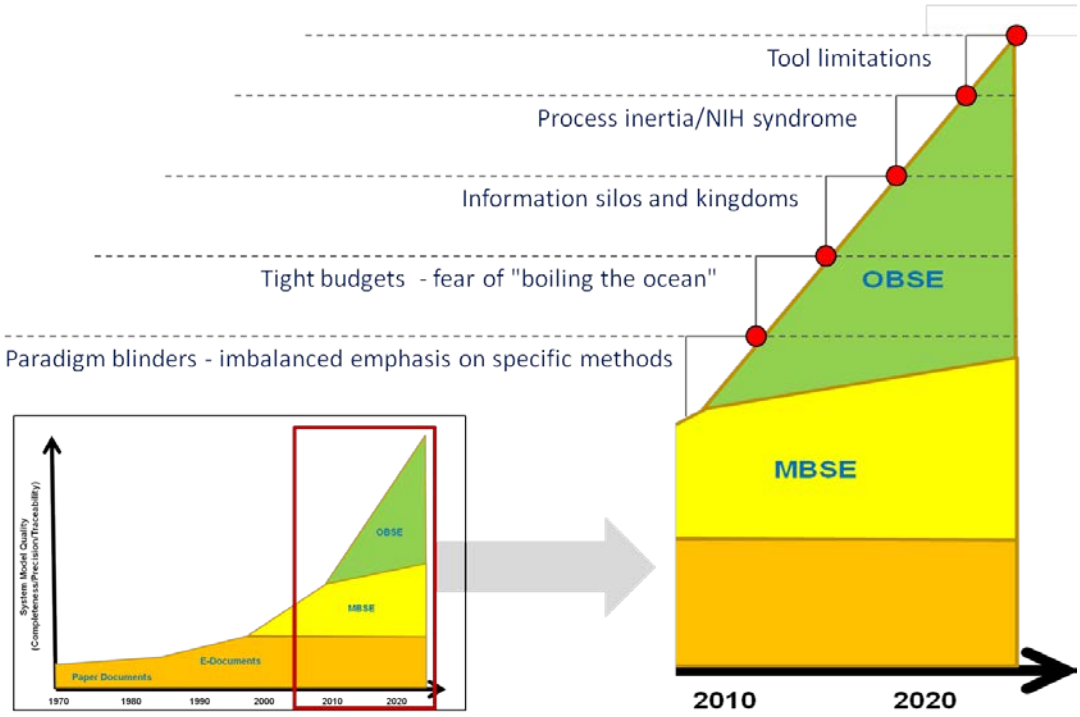
from random mutations introduced through the copy process, failure to synchronize multiple object copies when changes occur, and modification of objects to match a new context or presentation formats. Remove the temptation and ability to fudge the model for different audiences.

Accelerating the realization of OBSE

This paper proposes a shift in the focus of Systems Engineering from documents and views to objects. This movement is inevitable and underway, but its pace is inconsistent as roadblocks are encountered. Such roadblocks include:

- Process inertia/NIH syndrome
- Paradigm blinders - imbalanced emphasis on specific methods
- Tool limitations
- Information silos and kingdoms
- Tight budgets - fear of "boiling the ocean"

FIGURE 2: Roadblocks to OBSE realization



We don't propose to abandon documents or views, but to use them as collaboration media. In an Object-based Systems Engineering world, a document/view will never be the master. That role is played by the object database.

This paper does not propose a new Systems Engineering methodology, but rather a set of principles to integrate the information created by multiple, diverse methods. This paper won't propose the ultimate SE information architecture, but lays the foundation for development of such a model. The principles should be factored into the design of the Systems Engineering information architecture.

Object-based System Engineering Principles

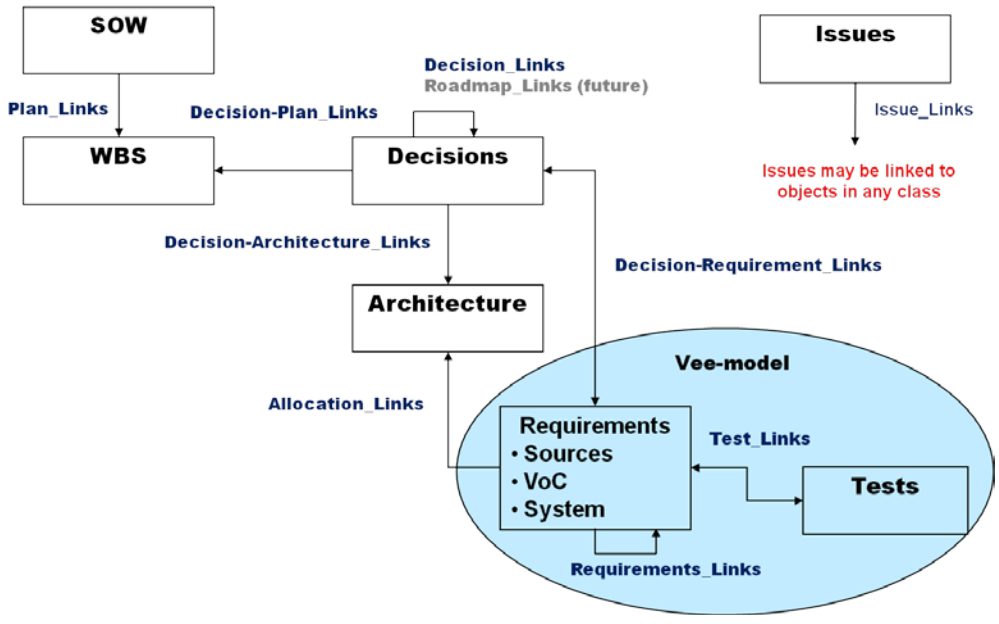
Object based Systems Engineering is based on the following principles.

1. ***Map all SE knowledge to object classes and subclasses***

   Create an initial information architecture model by answering the following questions:

   - What are the primary types of SE knowledge required to support the SE process?
   - Which classes and subclasses support the anticipated set of SE process use cases (types of systems/products to be developed; life cycle phase, project size, system context)?
   - How do these classes of information relate to one another? What classes of relationships connect various types of SE data?
   - What are the most vital and volatile classes of objects/relationships to preserve and maintain?

   FIGURE 3: OBSE Information Architecture



Ultimately, the value of any type of knowledge lies in the questions that it can answer. The knowledge to answer such questions may lie within a single class of objects or may be captured in the relationships between them.

FIGURE 4: Questions traceability can answer

SOW What is our scope & charter?	How will we accomplish our charter? Is our plan adequate?			**N-Squared Diagram Legend**	
How will work flow down to others?	**WBS** What's our plan? Who's responsible? Is plan adequate?			**Node A** / A-to-B interaction / B-to-A interaction / **Node B** — *Read the interactions clockwise*	
	How will we analyze or implement this decision?	**DECISIONS** Top N decisions? Status? Rationale? Consequences?	Why does this component exist? What role does it play?	Where did this requirement originate? Change impact?	
			ARCHITECTURE Components in our solution? Interfaces?		
		What decisions did this req't drive? Budget allocation? Change impact?	Allocated requirements? Budget flow-down?	**REQUIREMENTS** Success = ? Clear? Complete? Source?	Requirements per test? Verification coverage?
				Requirements met? Priority gaps to fix?	**TESTS** Test events/cases? Test enablers? Results/findings?

Test the initial information architecture against the most important questions that will be posed by the organization.

2. Refine this information architecture against multiple SE methods to make it as lean as possible (maximize cohesion, minimize coupling).

Information architecture is driven by the set of SE methods engines that have been selected to power the SE process. In order to refine the information architecture to make it scalable across a broader range of methods, ask the following:

- What methods could be used to create each class of object? What methods create the primary relationships among object classes?
- How would different methods engines change the information model (e.g. add new classes, attributes or relationships) This is the same thing as defining the derived requirements associated with each methods engine "alternative".
- Can an information model be created that captures the superset of all the classes/attributes/relationships required by the full range of methods engines under consideration?
- How can this model be made more lean; simplified to reduce the number of object classes and/or relationships?

For example, most defense projects use a requirements-centric Vee-model as the methods engine for their requirements development process. This method emphasizes requirement-to-requirement traceability down the left side of the Vee. If you switch to a decision-centric methods engine, a new type of relationship is required to capture decision-to-requirement traceability. [1]

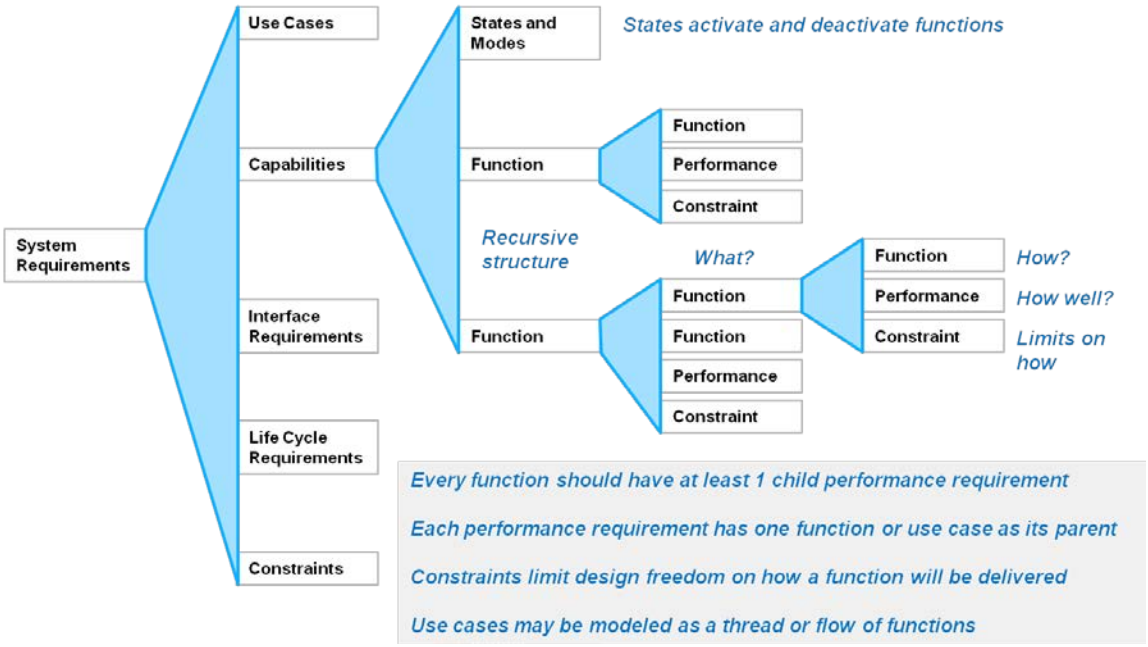
3. Create all objects in context (within a hierarchy appropriate to its class)

SE knowledge represents a network of associated objects. However, within each primary class of objects, a hierarchical structure (taxonomy) provides an efficient structure. These class hierarchies typically include subclasses arranged in a recursive pattern.

Class hierarchies are valuable knowledge patterns. They jump-start new projects by seeding the SE knowledge-base with a proven set of relevant objects. They highlight missing (but valuable) data as holes in the recursive structure. For example, a leaf-level functional requirement can be flagged as an incomplete branch in the requirements structure. Every functional requirement should have at least 1 performance requirement that specifies "How well?" the function must be performed.

SE knowledge objects never exist in a vacuum; they are always created in the context of other objects. There is a place for everything; the secret of OBSE is to create and keep everything in its place.

FIGURE 5: Requirement Hierarchy



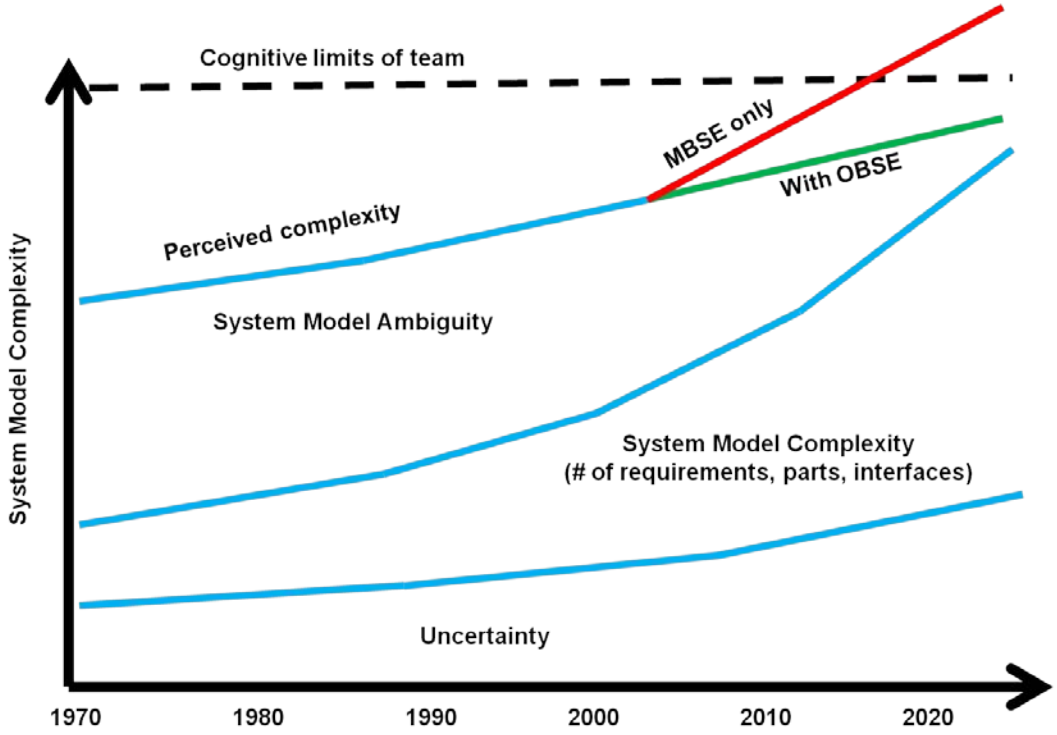
4. Define each object as a set of lean attributes and relationships (avoid free-form text).

The document model encourages contributors to write free-form text paragraphs. Even with guidance provided in the form of a document template, this approach leads to jumbled object, attribute and relationship data within paragraphs.

This individualized and situation-driven writing paradigm leaves translation of each paragraph into precise objects, attributes and relationships as an exercise for the individual reader. This is an ad hoc, non-repeatable process that contributes to the ambiguity in the system model.

As system complexity grows, so grows the need for precise capture of Systems Engineering knowledge as objects. Every system development project battles two fundamental enemies, uncertainty and ambiguity. These factors can combine to produce overwhelming complexity that leads to program failures.

FIGURE 6: Ambiguity vs. Uncertainty



Uncertainty is a product of the real-world unknowns and unknowable's. It is generally increasing as the pace of technology change/turnover increases. Uncertainty can be reduced through investments in knowledge-creating tasks (e.g. simulation and prototypes) but not driven to zero. Uncertainty may be managed, but can't be eliminated because there are no facts about the future.

Ambiguity is primarily a self-inflicted wound; it results from fuzzy and ad hoc methods that create high variance in the definition, context, derivation and interpretation of SE knowledge. The goal of OBSE is to drive ambiguity toward zero. This gives large and/or novel programs a fighting chance to keep the system model's perceived complexity within the cognitive limits of their team.

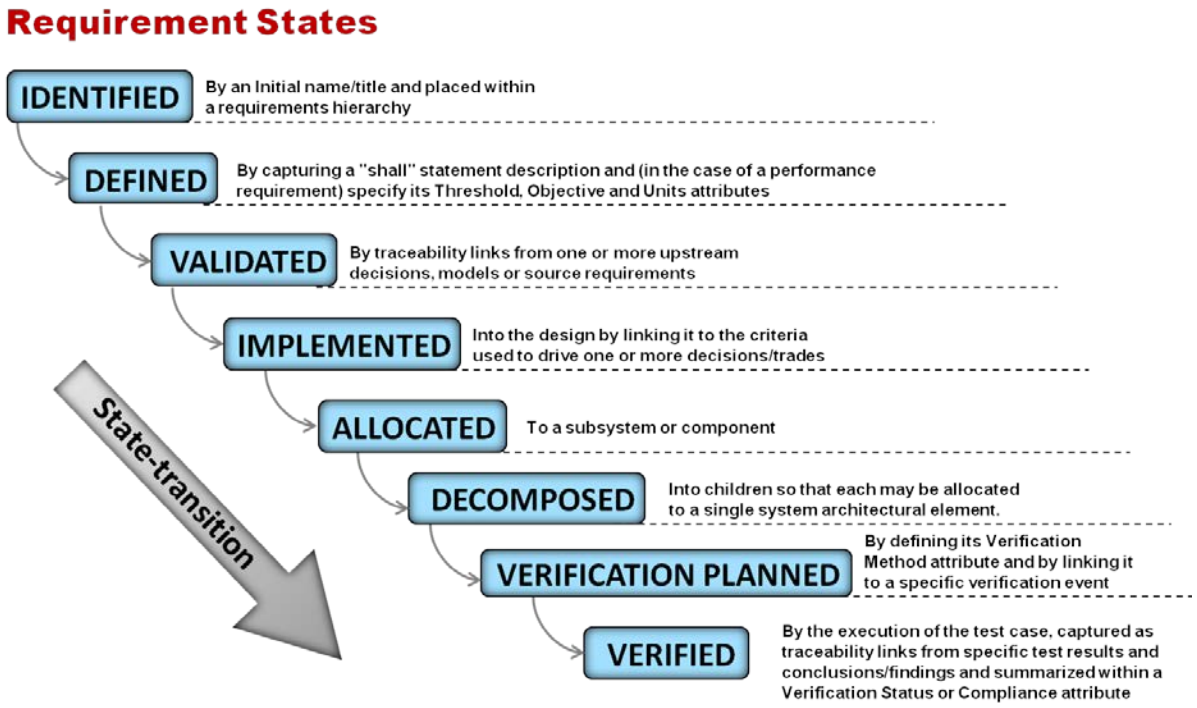
5. ***Strive for zero redundancy (store a single instance of an object; visualize in many ways).***

Systems Engineering projects suffer from another form of self-inflicted complexity. By creating copies of objects when populating various documents or viewing data from different contexts, the team grows the size of the system model. This larger-than-necessary model must be maintained and navigated, further reducing the team's efficiency.

The goal of OBSE is to maintain a single master instance of each object, i.e. maintain the leanest possible information model required to define the problem domain and to represent the proposed solution and the rationale behind it.

In OBSE, version control is focused on these master objects. Each class of object has its own life cycle model, i.e. it evolves through a series of states. Object-level versioning captures these states as changes to the attributes and relationships associated with each object.

FIGURE 7: Requirement States



In OBSE, documents and views become collaboration media, assembled automatically from the master objects (i.e. based on well-defined and published rules that ensure that the appropriate version/state of each object is included within each artifact).

6. Maintain continuous traceability as knowledge is derived.

If you don't maintain derivation traceability continuously, you will never have enough time to backfill it. It is very simple and cheap to capture at the time connections are made between objects; it becomes very expensive and impractical when attempted when en masse for a document or complex view.

Loss of this traceability multiplies the cost of proactive impact/change analysis (what-ifs) or renders it practically impossible where the original SMEs are unavailable or lack perfect recall.

7. Capture the precious and transient logic behind this knowledge derivation.

Derivation traceability is very precious, but transient knowledge. The scoring rationale behind a decision (each cell in an evaluation matrix) or allocation rationale between requirements and test cases is quickly forgotten and lost forever.

This data is more than a traceability link; it includes the derivation rationale, i.e. how or why A led to B+C+D. This is typically captured as an attribute on the link. A wise implementation of OBSE prompts users to succinctly capture this information as links are made.

To be complete, derivation rationale should also capture minority viewpoints and discussion threads associated with each object of interest. If the majority view doesn't lead to success, this data may be helpful in capturing lessons learned to drive process improvement.

8. Leverage the relationships between objects to proactively manage change.

If you maintain continuous derivation traceability, you can exploit it whenever any change is proposed to any part of the system model. This involves walking the linkages between objects and assessing the ripple effect, e.g. "How will the proposed change in this requirement affect the decisions (in which it was used as a criterion), architecture (to which it was allocated) or test case (to which it was assigned for verification)?

Understanding the ripple effect is typically a human-in-the-loop thought process, but it may be supplemented by simulation models.

9. Maintain continuity of objects across system/product life cycles and phases.

At the document and view level, very little useful information is passed between system/acquisition life cycle phases. Most programs operate as a fresh start or simply make a

copy of the documents/views and start modifying them. Continuity of thought depends on continuity of team members.

An OBSE database enables maximum reuse of relevant prior phase knowledge by maintaining continuity of objects. Instead of copies, the last state of each requirement, decision or test case is known and becomes the jumping-off point for next phase refinements or extensions.

10. Harvest and reuse knowledge patterns for each class of object

OBSE captures knowledge within class hierarchies and rule-based data structures. It is much easier to harvest these structures as knowledge patterns than to try to discern such patterns from ad hoc document prose.

These patterns enable knowledge reuse across many domains (project types, systems, technology families); this increases the ROI from investments in Systems Engineering discipline.

Benefits of Object based System Engineering

The principles of effective OBSE are not black-and-white or foolproof; they require skill, creativity and balanced judgment to apply to different types of systems, projects and SE environments. However, if skillfully applied, OBSE principles can yield many benefits:

- Simplify SE tasks. The leanest information model implies the leanest task model to populate and maintain it. OBSE highlights the value-added steps within Systems Engineering.

- Reduce overlapping efforts and information silos. Within a document-based SE process, key process areas tend to produce insulated artifacts. There is much redundant work and little connectivity between it. Decisions are disconnected from requirements and risks. TPM plans are unrelated to risks. Use cases are poorly linked to requirements. System architecture is isolated from the decisions that created it. These silos drive up costs and promote organizational politics, kingdom-building and turf-wars.

- Foster the insight that leads to innovation. The leanest possible information model reduces mental clutter and complexity overload to increase the likelihood of innovative insights. A class-based model encourages efficient, focused brainstorming. A shared information model fosters collaboration.

- Improve solution quality. Rule-based exception reports can highlight missing, but needed knowledge in any class of objects. These reports provide a list of loose ends to

resolve. Although this doesn't guarantee great thinking, it ensures completeness and consistency of Systems Engineering knowledge at the object level.

- Accelerate development. OBSE maximizes a team's ability to do parallel and aligned thinking. This can greatly shorten the time-to-capability to develop and deliver a complex system.

References

1. FITCH, J. Exploiting Decision-to-Requirements Traceability, briefing to NDIA CMMI Conference, November, 2009
2. CHHANIYARA, S; SAAJ, C; ALTHOFF-KOTZIAS, M; AHRNS, I; MAEDINGER, B; Model Based System Engineering for Space Robotics Systems, Sec. 2.2; ESA ASTRA 2011
3. ESTEFAN, J; Survey of Model-Based Systems Engineering (MBSE) Methodologies, Sec. 3.2; INCOSE MBSE Initiative, 2008
4. WALKER, M; The State of Model Based Systems Engineering, briefing to INCOSE Chesapeake Chapter, International Workshop, 2011